JUPYTER NOTEBOOKS

Jupyter Notebooks offer a great way to write and iterate on your Python code. It is an incredibly powerful tool for interactively developing and presenting data science projects. A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media. The intuitive workflow promotes iterative and rapid development, making notebooks an increasingly popular choice at the heart of contemporary data science, analysis, and increasingly science at large. Best of all, as part of the open source *Project Jupyter*, they are completely free.

Jupyter Notebooks is a web-based computing platform that allows developers to code, visualize, share and embed multimedia with explanatory text inside a single document. It is a famous tool for showcasing your work because You can see both the code and the results in the same file.

It is very famous among data scientists because you can get a better understanding of the data by executing each line of code separately. Jupyter Notebook allows its users to download the notebook in various file formats like PDF, HTML, Python, Markdown, or an. ipynb file.

Advantages of Jupyter Notebooks

Jupyter Notebooks offer several advantages for writing and interacting with Python code, making them a popular choice among data scientists, researchers, educators, and other professionals.

Let's take a quick overview of the program's main advantages.

i. **Interactive and Exploratory Environment**

My favorite aspect of Jupyter Notebooks is how easy it is to play with code within cells. By allowing users to write code in a cell-by-cell manner, you can quickly write code and test its output, allowing for a trial-and-error process that provides either instant gratification or instant correction.

Either way, you're able to write code at blazing speeds and ensure it accomplishes what you need it to.

ii. **Quick and Easy Visualizations**

Visualization is a major component of data science and analytics. Data is only as powerful as the story you can tell with it, so data workers need spaces to both analyze and visualize data. Jupyter Notebooks provide a platform to do both at once.

The integration of popular plotting libraries like Matplotlib, Seaborn, and Plotly within Jupyter Notebooks allows you to generate and visualize data plots directly alongside your code. This real-time visualization aids in understanding the data and the effects of your program.

I can't stress enough how important this is for data workers. Notebooks aren't just a tool — they're a *workbench*, providing a host of tools that make data analysis smooth and efficient.

iii. **Support for Collaboration**

Jupyter Notebooks can be shared easily with colleagues, collaborators, or the broader community. You can export notebooks in various formats, such as HTML, PDF, or Markdown, preserving the code, visualizations, and explanations.

I've found it easy to upload notebook documents to Github, allowing me to showcase my Python skills and invite collaboration from others.

Getting Started with Jupyter Notebooks!

Installation

Prerequisites

As you would have surmised from the above abstract we need to have Python installed on your machine. Either Python 2.7 or Python 3.+ will do.

Install Using Anaconda

The easiest way for a beginner to get started with Jupyter Notebooks is by installing it using Anaconda. Anaconda installs both Python3 and Jupyter and also includes quite a lot of packages.

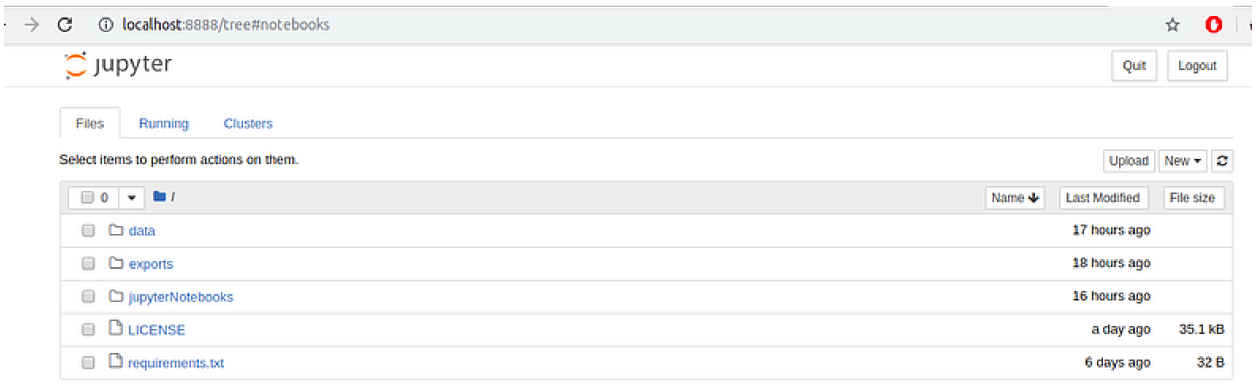
Install Using Pip

If for some reason, you decide not to use Anaconda, then you can install Jupyter manually using Python pip package, just follow the below code:

Launching First Notebook

To launch a Jupyter notebook, open your terminal and navigate to the directory where you would like to save your notebook. Then type the below command and the program will instantiate a local server at http://localhost:8888/tree.

A browser window should immediately pop up with the Jupyter Notebook interface. As you might have already noticed Jupyter's Notebooks and dashboard are web apps, and Jupyter starts up a local Python server to serve these apps to your web browser. It makes Jupyter Notebooks platform independent and thus making it easier to share with others.

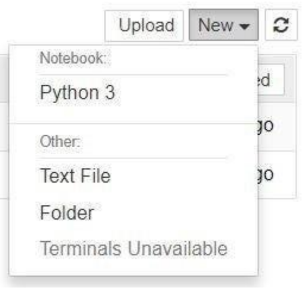


Select items to perform actions on them.

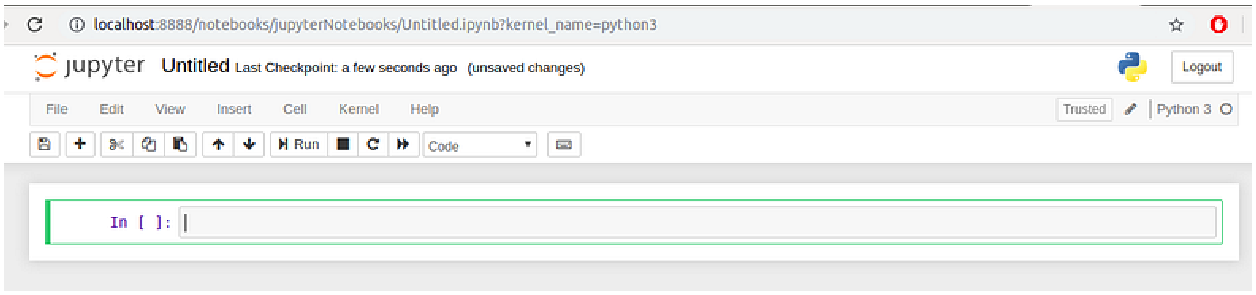
Name	Last Modified	File size
data	17 hours ago	
exports	18 hours ago	
jupyterNotebooks	16 hours ago	
LICENSE	a day ago	35.1 kB
requirements.txt	6 days ago	32 B

The **Files** tab is where all your files are kept, the **Running** tab keeps track of all your processes and the third tab, **Clusters**, is provided by IPython parallel, IPython's parallel computing framework. It allows you to control many individual engines, which are an extended version of the IPython kernel.

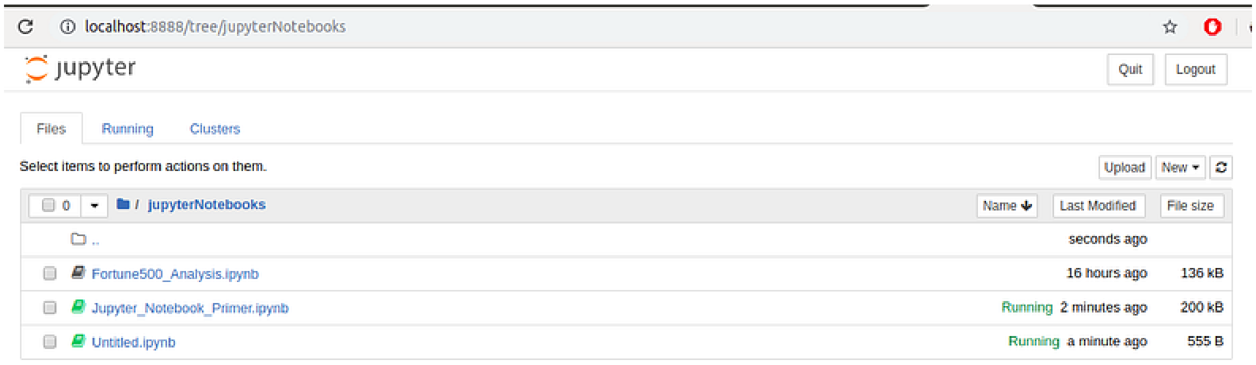
Let's start by making a new notebook. We can easily do this by clicking on the **New** drop-down list in the top- right corner of the dashboard. You see that you have the option to make a Python 3 notebook as well as a regular text file, a folder, and a terminal. Please select the Python 3 notebook option.



Your Jupyter Notebook will open in a new tab as shown in the below image.


localhost:8888/notebooks/jupyterNotebooks/Untitled.ipynb?kernel_name=python3
jupyter
Untitled Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Trusted
Python 3
Run
Code
In [ ]:

Now each notebook uses its own tab so that you can open multiple notebooks simultaneously. If you switch back to the dashboard, you will see the new file **Untitled.ipynb** and you should see some green text that tells you your notebook is running.



Select items to perform actions on them.

Name	Last Modified	File size
/ jupyterNotebooks		
..	seconds ago	
Fortune500_Analysis.ipynb	16 hours ago	136 kB
Jupyter_Notebook_Primer.ipynb	Running 2 minutes ago	200 kB
Untitled.ipynb	Running a minute ago	555 B

Why a .ipynb file?

.ipynb is the standard file format for storing Jupyter Notebooks, hence the file name **Untitled.ipynb**. Let's begin by first understanding what an *.ipynb* file is and what it might contain. Each *.ipynb* file is a text file that describes the contents of your notebook in a JSON format. Each cell and its contents, whether it be text, code or image attachments that have been converted into strings of text, is listed therein along with some additional metadata. You can edit the metadata yourself by selecting "Edit > Edit Notebook Metadata" from the menu bar in the notebook.

You can also view the contents of your notebook files by selecting "Edit" from the controls on the dashboard, there's no reason to do so unless you really want to edit the file manually.

Understanding the Notebook Interface

Now that you have an open notebook in front of you take a look around. Check out the menus to see what the different options and functions are readily available, especially take some time out to scroll through the list of commands in the command palette, the small button with the keyboard icon (or just press Ctrl + Shift + P)

There are two prominent terminologies that you should care to learn about: **cells** and **kernels** are key both to understanding Jupyter and to what makes it more than just a content writing tool. Fortunately, these concepts are not difficult to understand.

- A **kernel** is a program that interprets and executes the user's code. The Jupyter Notebook App has an inbuilt kernel for Python code, but there are also kernels available for other programming languages.
- A **cell** is a container for text to be displayed in the notebook or code to be executed by the notebook's kernel.

Cells

Cells from the body of a notebook. In the screenshot for a new notebook(Untitled.ipynb) in the section above, the box with the green outline is an empty cell. There are 4 types of cells:

- **Code** — This is where you type your code and when executed the kernel will display its output below the cell.
- **Markdown** — This is where you type your text formatted using Markdown and the output is displayed in place when it is run.
- **Raw NBConvert** — It's a command line tool to convert your notebook into another format (like HTML, PDF, etc.)
- **Heading** — This is where you add Headings to separate sections and make your notebook look tidy and neat. This has now been merged into the Markdown option itself. Adding a '#' at the beginning ensures that whatever you type after that will be taken as a heading.

Let's test out how the cells work with a classic hello world example. Type print('Hello World!') into the cell and click the **Run button** in the toolbar above or press Ctrl + Enter.

Hello World!

When you run the cell, its output will is also displayed below and the label to its left changes from **In[ ]**

to **In[1]** . Moreover, to signify that the cell is still running, Jupyter changes the label to **In[*]**

Additionally, it is important to note that the output of a code cell comes from any text data specifically printed during the execution of the cell, as well as the value of the last line in the cell, be it alone variable, a function call, or something else.

Markdown

Markdown is a lightweight, markup language for formatting plain text. Its syntax has a one-to-one correspondence with HTML tags. As this article has been written in a Jupyter notebook, all of the narrative text and images you can see are achieved in Markdown. Let's cover the basics with a quick example.

When attaching images, you have three options:

- Use a URL to an image on the web.
- Use a local URL to an image that you will be kept alongside your notebook, such as in the same git repo.
- Add an attachment via "Edit > Insert Image"; this will convert the image into a string and store it inside your notebook .ipynb file.

Note that adding an image as an attachment will make the .ipynb file much larger because it is stored inside the notebook in a string format.

There are a lot more features available in Markdown. Once you have familiarized yourself with the basics above, you can refer to the official guide from the creator, **John Gruber**, on his website.

Kernels

Behind every notebook runs a kernel. When you run a code cell, that code is executed within the kernel and any output is returned back to the cell to be displayed. The kernel's state persists over time and between cells — it pertains to the document as a whole and not individual cells.

For example, if you import libraries or declare variables in one cell, they will be available in another. In this way, you can think of a notebook document as being somewhat comparable to a script file, except that it is multimedia. Let's try to understand this with the help of an example. First, we'll import a Python package and define a function.

Once we've executed the cell above, we can reference os, binascii and sum in any other cell.

The output should look something like this:

Majority of times, the flow in your notebook will be top-to-bottom, but it's common to go back to make changes. In this case, the order of execution is stated to the left of each cell, such as In [2], will let you know whether any of your cells have stale output. And if you ever wish to reset things, there are several incredibly useful options from the Kernel menu:

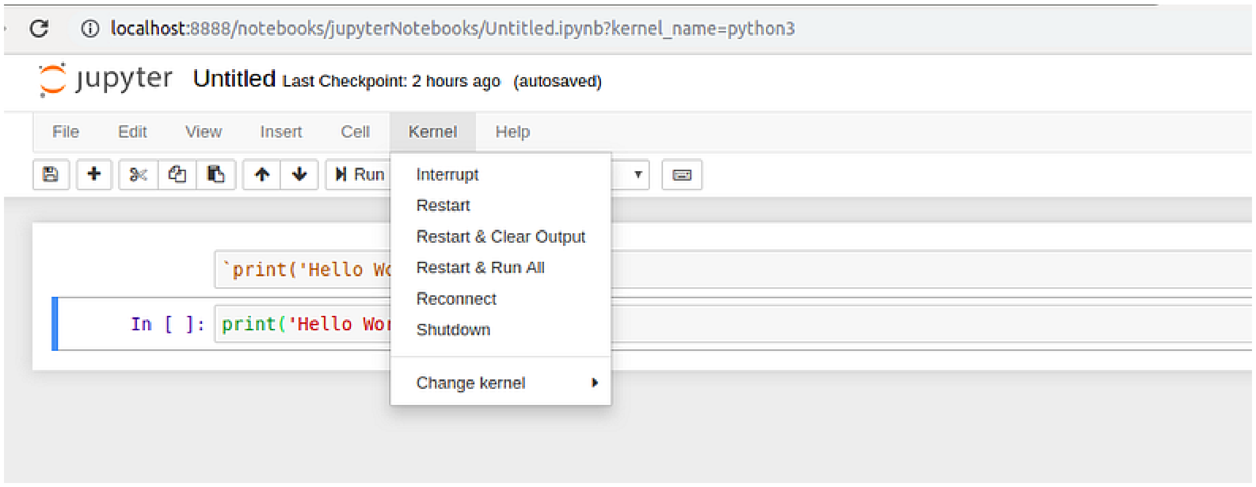
- **Restart**: restarts the kernel, thus clearing all the variables, etc that were defined.
- **Restart & Clear Output**: same as above but will also wipe the output displayed below your code cells.
- **Restart & Run All**: same as above but will also run all your cells in order from first to last.
- **Interrupt**: If your kernel is ever stuck on computation and you wish to stop it, you can choose the Interrupt option.

Naming Your Notebooks

It is always a best practice to give a meaningful name to your notebooks. It may appear confusing, but you cannot name or rename your notebooks from the notebook app itself. You must use either the dashboard or your file browser to rename the *.ipynb* file. We'll head back to the dashboard to rename the file we created earlier, which will have the default notebook file name Untitled.ipynb.

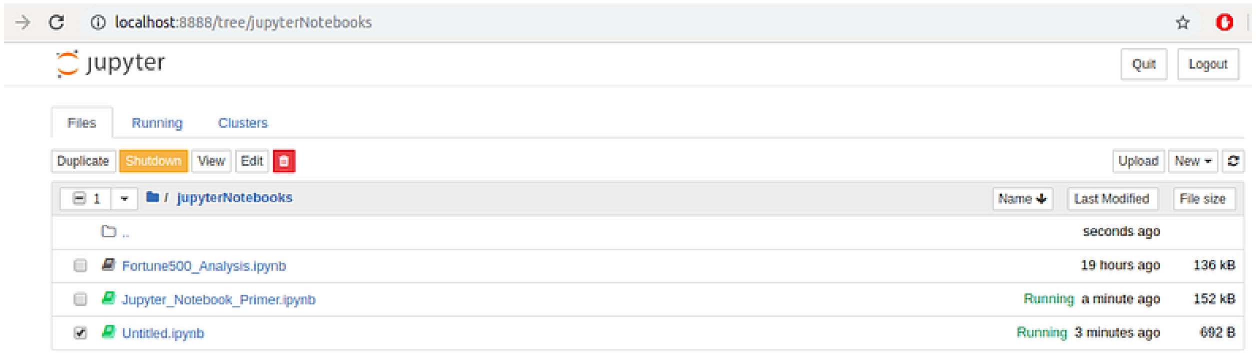
We cannot rename a notebook while it is running, so let's first shut it down. The easiest way to do this is to select "File > Close and Halt" from the notebook menu. However, we can also shut down the kernel either by going to "Kernel > Shutdown" from within the notebook app or by selecting the notebook in the dashboard and clicking "Shutdown" (see images below).

Shutdown the kernel from Notebook App:


localhost:8888/notebooks/jupyterNotebooks/Untitled.ipynb?kernel_name=python3
jupyter Untitled Last Checkpoint: 2 hours ago (autosaved)
File
Edit
View
Insert
Cell
Kernel
Help
Run
Interrupt
Restart
Restart & Clear Output
Restart & Run All
Reconnect
Shutdown
Change kernel

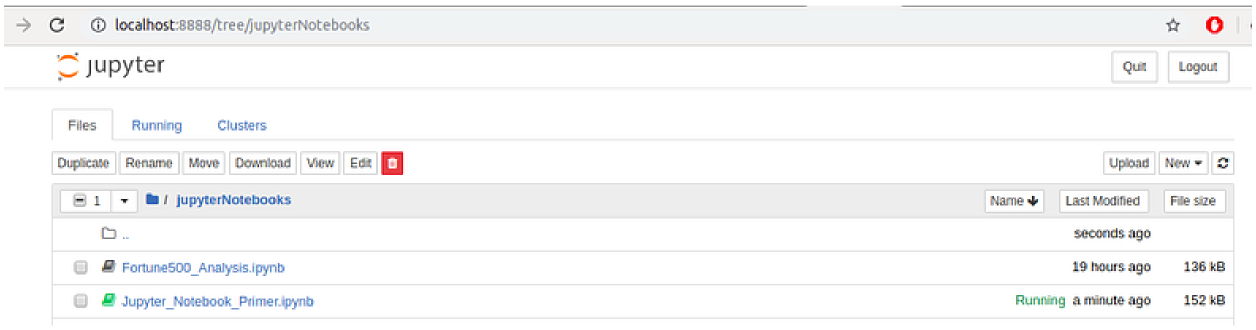
```
`print('Hello Wo
In [ ]: print('Hello Wo
```

Shutdown the kernel from Dashboard:



..		seconds ago	
Fortune500_Analysis.ipynb		19 hours ago	136 kB
Jupyter_Notebook_Primer.ipynb	Running	a minute ago	152 kB
Untitled.ipynb	Running	3 minutes ago	692 B

Once the kernel has been shut down, you can then select your notebook and click "Rename" in the dashboard controls.



Sharing Your Notebooks

When we talk about sharing a notebook, there are two things that might come to our mind. In most cases, we would want to share the end-result of the work, i.e. sharing non-interactive, pre-rendered version of the notebook, very much similar to this article; however, in some cases we might want to share the code and collaborate with others on notebooks with the aid of version control systems such as Git which is also possible.

Before You Start Sharing

A shared notebook will appear exactly in the state it was in when you export or save it, including the output of any code cells. Therefore, to ensure that your notebook is share-ready, so to speak, there are a few steps you should take before sharing:

1. Click "Cell > All Output > Clear"
2. Click "Kernel > Restart & Run All"
3. Wait for your code cells to finish executing and check they did so as expected

This will ensure your notebooks don't contain intermediary output, have a stale state, and executed in order at the time of sharing.

Exporting Your Notebooks

Jupyter has built-in support for exporting to HTML, Markdown and PDF as well as several other formats, which you can find from the menu under "File > Download as". It is a very convenient way to share the results with others. But if sharing exported files doesn't cut it for you, there are also some immensely popular methods of sharing .ipynb files more directly on the web.

GitHub

- With home to over 2 million notebooks, GitHub is the most popular place for sharing Jupyter projects with the world. GitHub has integrated support for rendering .ipynb files directly both in repositories and gists on its website.
- You can just follow the GitHub guides for you to get started on your own.

Nbviewer

- NBViewer is one of the most popular notebook renderers on the web.

- If you already have somewhere to host your Jupyter Notebooks online, be it GitHub or elsewhere, NBViewer will render your notebook and provide a shareable URL along with it.

MACHINE LEARNING LIBRARIES

Libraries provide specific functionalities, while frameworks offer a complete set of tools for developing a fully-fledged application. So when designing a software solution, you might use many libraries, but typically only one or a few frameworks. A library is a collection of prewritten codes, predefined methods, and classes that programmers can use to simplify and accelerate development and solve a specific problem. It includes functions, class definitions, important constants, etc. As a result, you can skip writing code to achieve specific features.

Machine learning libraries act as a bridge between complex mathematical algorithms and the practical implementation of machine learning models. They provide a set of pre-built functions and tools that simplify the process of developing and deploying machine learning solutions. By using these libraries, developers and data scientists can focus more on the problem-solving aspect rather than spending time on coding complex algorithms from scratch.

Popular ML libraries include:

a. Scikit-learn

Scikit-learn is arguably one of the most popular machine-learning libraries in Python. It provides a comprehensive set of tools for data preprocessing, feature selection, model training, and evaluation. Scikit-learn supports a wide range of machine learning algorithms, including linear regression, decision trees, support vector machines, and neural networks.

With Scikit-learn, you can easily handle tasks such as data splitting, cross-validation, hyperparameter tuning, and model evaluation. Its clean and intuitive API makes it a favorite choice for beginners and seasoned data scientists alike.

What is it: Scikit-learn is a Python library for implementing machine learning algorithms. A developer named David Cournapeau originally released scikit-learn as a student in 2007. The open source community quickly adopted it and has updated it numerous times over the years [2].

- Scikit-learn includes every core machine learning algorithm, among them vector machines, random forests, gradient boosting, k-means clustering, and DBSCAN.
- It was designed to work seamlessly with NumPy and SciPy (both described below) for data cleaning, preparation, and calculation.
- It has modules for loading data as well as splitting it into training and test sets.
- It supports feature extraction for text and image data.

b. NumPy

What is it: NumPy is a Python package for working with arrays, or large collections of homogenous data. You can think of an array like a spreadsheet, where numbers are stored in columns and rows. NumPy is an open-source Python library for arrays processing. It can execute algebraic, logical, and statistical operations over matrices and multidimensional arrays. The NumPy library is among the most popular Python tools for AI and data science computing [4].

NumPy stands for Numerical Python. As the name suggests, it is a library primarily intended for calculations. With it, you can save a lot of time when performing complex matrix operations.

Background: Python wasn't originally intended for numerical computation when it was launched in 1991. Still, its ease of use caught the scientific community's attention early on. Over the years, the open source community developed a succession of packages for numerical computing. In 2005, developer Travis Oliphant combined over a decade's worth of open-source developments into a single library for numerical computation, which he called NumPy

Features: The core feature of NumPy is support for arrays, which allows you to quickly process and manipulate large collections of data.

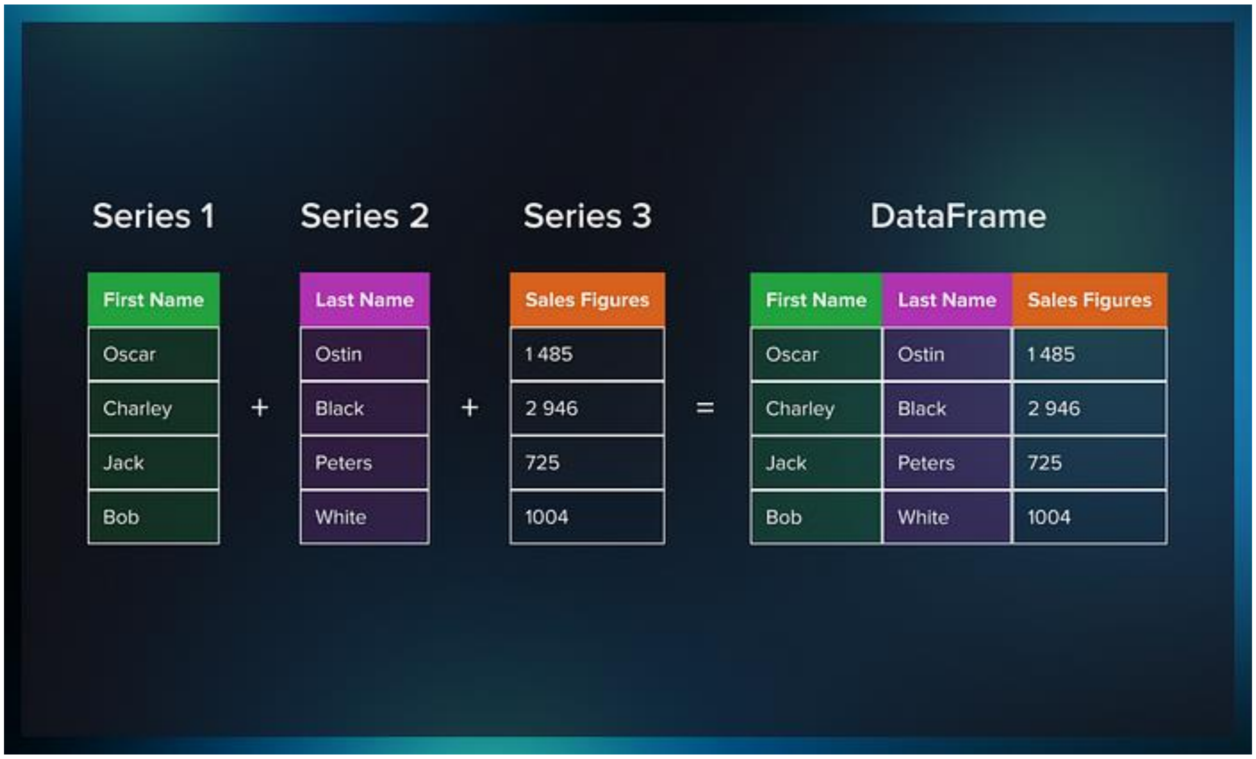
- Arrays in NumPy can be n-dimensional. This means the data can be a single column of numbers, or many columns and rows of numbers.
- NumPy has modules for performing some linear algebra functions.
- It also has modules for graphing and plotting numerical arrays.
- Data in NumPy arrays is homogenous, which means it must all be defined as the same type (numbers, strings, Boolean values, etc.). This means data gets processed efficiently.

c. Pandas

What is it: Pandas is a package for simultaneously working with different types of labeled data. You'd use it, for example, if you need to analyze a CSV file containing numerical, alphabetical, and string data. Pandas is the best option for handling tabular data and time series. This open-source library has a comprehensive list of built-in commands that save ML developers the need to write code specifically for certain mathematical operations. In addition to data manipulation, pandas also support data transformation and visualization.

The library uses two main data structure types:

- Series (a 1-dimensional labeled data array).
- DataFrame (a 2-dimensional data structure)



First Name
Oscar
Charley
Jack
Bob

Last Name
Ostin
Black
Peters
White

Sales Figures
1 485
2 946
725
1004

First Name	Last Name	Sales Figures
Oscar	Ostin	1 485
Charley	Black	2 946
Jack	Peters	725
Bob	White	1004

Background: Wes McKinney released Pandas in 2008. It builds on NumPy (and, in fact, you must have NumPy installed to use Pandas) and extends that package to work with heterogeneous data.

Features: The core feature of Pandas is its variety of data structures, which let users perform an assortment of analysis operations.

- Pandas has a variety of modules for data manipulation, including reshape, join, merge, and pivot.
- Pandas has data visualization capabilities.
- Users can perform mathematical operations including calculus and statistics without calling on outside libraries.
- It has modules that help you work around missing data.

e. SciPy

What is it: SciPy is a Python library for scientific computing. It contains packages and modules for performing calculations that help scientists conduct or analyze experiments.

Background: In the late 1990s and early 2000s, the Python open-source community began working on a collection of tools to meet the needs of the scientific community. In 2001, they released these tools as SciPy. The community remains active and is always updating and adding new features.

Features: SciPy's packages comprise a complete toolkit of mathematical techniques from calculus, linear algebra, statistics, probabilities, and more.

- Some of its most popular packages for data scientists are for interpolation, K-means testing, numerical integration, Fourier transforms, orthogonal distance regression, and optimization.
- SciPy also includes packages for image processing and signal processing.
- The Weave feature allows users to write code in C/C++ within Python.